

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
9 January 2003 (09.01.2003)

PCT

(10) International Publication Number
WO 03/003232 A2

(51) International Patent Classification⁷: **G06F 15/16**

(21) International Application Number: PCT/IB02/02417

(22) International Filing Date: 20 June 2002 (20.06.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
01202517.7 29 June 2001 (29.06.2001) EP

(71) Applicant (for all designated States except US): **KONINKLIJKE PHILIPS ELECTRONICS N.V.** [NL/NL];
Groenewoudseweg 1, NL-5621 BA Eindhoven (NL).

(72) Inventors; and

(75) Inventors/Applicants (for US only): **NIEUWLAND, Andre, K.** [NL/NL]; Internationaal Octrooibureau B.V., Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL). **LIPPENS,**

Paul, E., R. [NL/NL]; Internationaal Octrooibureau B.V., Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL). **GANGWAL, Om, P.** [IN/NL]; Internationaal Octrooibureau B.V., Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL).

(74) Agent: **DE JONG, Durk, J.**; Internationaal Octrooibureau B.V., Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL).

(81) Designated States (*national*): CN, JP, US.

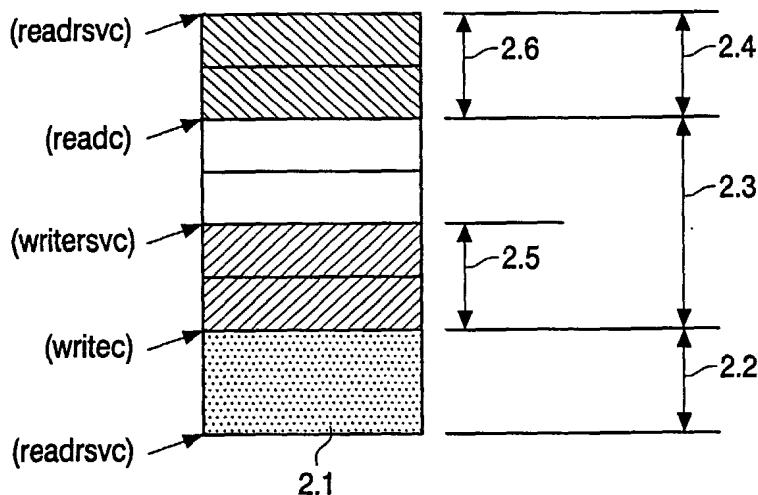
(84) Designated States (*regional*): European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR).

Published:

— without international search report and to be republished upon receipt of that report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: DATA PROCESSING APPARATUS AND A METHOD OF SYNCHRONIZING A FIRST AND A SECOND PROCESSING MEANS IN A DATA PROCESSING APPARATUS



(57) Abstract: A data processing apparatus according to the invention comprising at least a first (1.2) and a second processing means (1.3). The first processing means being capable of providing data by writing tokens in a buffer means (2.1) which are readable by the second processing means for further processing. The processing means are assigned a first and a second synchronization counter. The first synchronization counter (writec) is modifiable by the first processing means (2.1) and is readable by the second processing means (1.3). The second synchronization counter (readc) is modifiable by the second processing means (1.3), and readable by the first processing means (1.2). The first synchronization counter (writec)

is indicative for a number of tokens being written by the first processing means (1.2). The second synchronization counter (readc) is indicative for a number of tokens being read by the second processing means (1.3).



WO 03/003232 A2

Data processing apparatus and a method of synchronizing a first and a second processing means in a data processing apparatus

The invention relates to a data processing apparatus comprising at least a first and a second processing means, the first processing means being capable of providing data by writing tokens in a buffer means which are readable by the second processing means for further processing, the processing means being assigned a first and a second synchronization indicator, the first synchronization indicator being modifiable by the first processing means and being readable by the second processing means, and the second synchronization indicator being modifiable by the second processing means, and readable by the first processing means.

The invention further relates to a method of synchronizing a first and a second processing means in a data processing apparatus, by means of a first and a second synchronization indicator, the first processing means being capable of providing data by writing tokens via a buffer means to the second processing means for further processing, wherein

- a. the first processing means generates one or more tokens,
- b. compares the first and the second synchronization indicator,
- c. in dependence of this comparison either
 - c1. writes the tokens to the buffer means and modifies the first synchronization indicator, after which it continues with step a
 - c2. repeats steps b-c, and wherein
- d. the second processing means compares the first and the second synchronization indicator,
- e. in dependence of this comparison either
 - e1. reads the tokens from the buffer means and modifies the second synchronization indicator,
 - e2. repeats steps d,e.

Signal-processing functions determine the performance requirements for many products based on standards like MPEG-x, DVB, DAB, and UMTS. This calls for efficient implementations of the signal processing components of these products. However, because of

evolving standards and changing market requirements, the implementation requires flexibility and scalability as well. A macropipeline setup is a natural way to model these applications, since streams of data are processed; in this setup the functions (tasks) are the stages and there are buffers between the stages to form the pipeline. This is a way to exploit task-level

5 parallelism (TLP), because all stages can operate in parallel.

A data processing apparatus and method according to the introductory paragraphs are known from WO 99/22296. In this description and the accompanying claims a token will indicate a unit within the buffer means that may contain an arbitrary number of data elements. A unit may contain for example a single sample of an audio stream, or e.g. a
10 complete matrix of picture elements (pixels) of a video stream. The known dataprocessing apparatus comprises a first station which produces tokens in the form of blocks of data, and a second station which consumes the tokens. The first station has control over a first synchronization indicator, in the form of a production semaphore, and the second station has control over a second synchronization indicator in the form of a consumption semaphore.

15 After having produced a token, the first station checks whether the value of the semaphore of the second station and possible other consuming stations have the same value as its own semaphore. If this is the case it loads the produced token in a buffer and toggles the production semaphore. The consuming station waits until the production semaphore differs from its own semaphore before it reads the token from the buffer. Each token has its own
20 semaphore. The known data processing apparatus avoids write conflicts with respect to a semaphore, because each semaphore can only be written by one station. Note that step a might be postponed until step c1, that is: first is checked whether the buffer means contains space for writing tokens, thereafter data is produced and or written to the buffer means. The buffer means are for example a shift register, or another kind of memory wherein the tokens
25 can be stored sequentially.

It is a purpose of the invention to provide a data processing apparatus which has an improved efficiency. According to the invention the data processing apparatus is therefore characterized in that each of the synchronization indicators is represented by a counter, the counter (p-counter) which represents the first synchronization indicator being
30 indicative for a number of tokens written by the first processing means, and the counter which represents the second synchronization indicator (c-counter) being representative for an amount of tokens read by the second processing means from the buffer means. The first and the second processing means are each able to determine the amount of available full/empty

tokens from the difference between the counters in relation with the maximum number of tokens.

For example if the buffer means has a capacity of $Nb=16$ tokens, and the p-counter has a value Np of 8 tokens and the c-counter a value Nc of 5 tokens then

5 $Np - Nc \pmod{Nb} = 3$ tokens are available to be read by the second processing means, and $Nc - Np \pmod{Nb} = 13$ tokens are available to be written by the first processing means.

10 The present data processing apparatus is an improvement over the known apparatus in that it replaces the individual semaphores of the first processing means by a single counter, and replaces the individual semaphores of the second processing means by another counter.

Instead of toggling the production semaphore, the producer now increments the counter representing the first synchronization indicator (hereinafter also called production counter) after having provided a token to the buffer means. Likewise, the consumer
15 increments the counter representing the second synchronization semaphore (consumption counter) if it has consumed a token.

Read write hazards are prevented because each of the counters are only modified by one of the processing means.

20 In order to determine whether a token is available for writing (empty), the producer subtracts the consumption counter from the production counter, and compares the result with the maximum number of tokens which can be stored in the buffer. Likewise the consumer subtracts the two counters to check whether filled tokens are available for reading.

It is noted that US 6,173,307 B1 discloses a multiprocessor system comprising circular queue shared by multiple producers and multiple consumers. Any producer or
25 consumer can be permitted to preempt any producer or consumer at any time without interfering with the correctness of the queue.

It is further noted that US 4,916,658 describes an apparatus comprising a dynamically controlled buffer. The buffer is suitable for storing data words consisting of several storage locations together with circuitry providing a first indicator that designates the
30 next storage location to be stored into, a second indicator designating the next storage location to be retrieved from, and circuitry that provides the number of locations available for storage and the number of locations available for retrieval.

In a preferred embodiment of the inventive data processing apparatus at least one of the processors locally maintains a reservation counter, wherein a first command issued

by the at least one processor results in a verification whether a requested number (one or more) of tokens is available to it by comparing its reservation counter with the synchronization counter maintained by the other processor, the reservation counter being updated if the outcome of the verification is affirmative.

5 In this way the processor can reserve a zone within the buffer having a lower boundary referred to by the synchronization counter and a lower boundary referred to by the reservation counter. Within this zone it may randomly access the buffer. Preferably the reservation counter and the synchronization counter are stored in a single dataword so as to enable a quick access of this information.

10 A preferred embodiment of the data processing apparatus according to the invention is characterized in that the counters have a range which is larger than the total number of data elements which is contained in a full buffer means. In this way it is prevented that ambiguity may arise when the counters are equal. This is illustrated by the following example. Note that when the p-counter is incremented from 0 to 1, it means that a first token
15 has been written. After reading a block, c-counter is also incremented from 0 to 1, meaning the buffer is empty again. Assume that the buffer means has a capacity of $N_b=16$ tokens, and that the counters count from 0 to 16. Initially, $N_p = N_c = 0$. Then when the p-count attains the value 15 the buffer-means is full. Wrapping back the p-count to 0 before the second processing means, the consumer, has read a token would result in the erroneous situation that
20 the buffer is full, while the counters are equal. By using a counters having range larger than the number of data elements this ambiguity is avoided.

A preferred embodiment of the data processing apparatus according to the invention is characterized in that the counters have one extra bit. In this way the range of the counters is twice the maximum number of tokens. By definition: if the two counters are
25 equal, including the extra bit then all tokens are empty. If only the extra bits of the counters differ then all tokens in the buffer are full. Note, that it is also possible to have an alternative definition: When the counters are equal and the extra bits are different then the buffer is empty, and when the extra bits are equal than the buffer is full. Important is that the meaning of the bits is defined, and that full is distinguishable from empty.

30 The processing means may be implemented either as hardware dedicated to a particular task, or as a software program executed by a general purpose processor or as a combination of hardware and software.

In an embodiment the first processing means get access to an empty token by means of a producer-claim call, and releases the filled token with a producer-release call. The

call to the producer-claim subroutine results in that the difference between the production counter and the consumption counter is determined, and dependent on the result the first processing means is given access to the buffer means. The producer release call has the result that the production counter is incremented, meaning that the claimed token is filled with data
5 produced by the first processing means.

In an embodiment the second processing means get access to a filled token, with a consumer-claim call, and releases the token with a consumer-release call. The token may be emptied or have a changed content. The subroutine activated by the consumer-claim call determines the difference between the production counter and the consumption counter,
10 and dependent on the result the second processing means is given access to the buffer means. The call to the consumer release subroutine has the effect that the consumption counter is incremented, meaning that the data corresponding to a claimed full token (which was produced by the first processing means) has been (marked as) read.

In a variation of said embodiment the first producing means may get access to
15 a plurality of tokens by means of the producer-claim call. Likewise the consumer may get access to a plurality of tokens by means of the consumer-claim call.

If multiple tokens are claimed but not directly released, the number of claimed tokens should be tracked, for example by a claim counter. The claim counter of the first processing means need not be readable for the second processing means and vice versa. A
20 claim counter for example of the first processing means, may be implemented as a separate counter, but otherwise be merged with e.g. the production counter.

The counters may be implemented as a part of a datastructure belonging to a token buffer (communication channel).

In a preferred embodiment a local copy of the datastructure is maintained near
25 the processing means, e.g. by caching. This reduces the latency in accessing the datastructure and therewith the load on the communication network.

When a claim for a token fails, there is no full token (consumer) or empty token (producer) available. In that case, the processing means has several options, e.g. execute another task or wait. When waiting the second processing means could poll (un-
30 cached) the production-counter of the first processing means. A waiting first processing means could poll (un-cached) the consumption-counter. A waiting processing means could also fall back to an idle wait mode, e.g. a sleep mode. This may have the advantage of a low energy use. To wake a processing means from the sleep mode, a signal, e.g. memory mapped, could be sent as a wake-up signal after the appropriate counter has been updated. During

normal operation these signals may be ignored since they are only required as a wake-up signal.

An embodiment of a data processing apparatus according to the invention is characterized in that the first processing means comprises a register means for locally storing an indication of the amount of tokens available to be written. After the first processing means has calculated the amount of tokens available to be written from the p-counter, the c-counter and the number of tokens available in the buffer means, it may store this amount in the register means. As soon as it has written one or more tokens, it updates the p-counter and the register means. The register means give a pessimistic estimation of the amount of tokens available for writing, which is save: As a token is written by the first processing means the value in the register means is decreased, and if a token is read by the second processing means, and therewith becomes available it remains unaltered. Therefore, as long as the register indicates that tokens are available for writing the first processing means may continue to do so. In the meantime the first processing means do not have to check the c-counter, which saves busload. Only if the register means indicates that no tokens are available anymore for writing, the first processing means has to access the c-counter and redo the calculation.

An embodiment of the data processing apparatus according to the invention, is characterized in that the second processing means comprises a register means for locally storing an indication of the amount of tokens available to be read. In a way analog to what was described above, this reduces access of the second processing means to the p-counter.

In an embodiment wherein one of the processing means is implemented by hardware dedicated to a particular task, and another of the processing means is implemented as a software program executed by a general purpose processor the synchronization information is preferably distributed over a shell and the shared memory.

A shell coupled to a processor facilitates multitasking, in that it may reduce the number of interrupts which has to be handled by the processor itself. This reduces the number of times that an idle processor has to be activated unnecessarily, or that a processor has to interrupt an other task which it is processing. In this way the efficiency of the processor is improved.

Several options are possible to implement such a shell for selecting interrupt signals depending on the type of interrupt signals and how they are encoded. In an embodiment the interrupt signals are indicative for a data channel of the processor. A practical way to implement this is by assigning bits in a register to respective input channels of

the processor. For example a 32 bit register could support 32 input channels, wherein for example channel 0 is assigned bit 0, channel 1 is assigned bit 1 of the register etc. When an other processor sends an interrupt signal destined for channel **k** of the processor the corresponding bit **k** of the signal register is set. The shell of the receiving processor can select specific interrupt signals by means of a mask register, wherein each bit represents for a particular channel whether the processor wants to ignore the interrupt or not. E.g. if the bit corresponding to channel two is masked, this signal will not cause an interrupt in the processor, and no wake up will happen. In this example, the processor could be busy with processing, in which all bits will be masked, or the processor /task could be waiting for a full/empty token on channel 1, in which case it is not interested on what happens on channel 2.

The signal and the mask register could have an arbitrary number of bits depending on the number of channel which should be supported. Alternatively it is possible to support each channel by a number and use a list or look-up table to determine whether the processor should be interrupted for that channel or not. This is however a more complex solution.

Instead of identifying a interrupt signal by its channel number it could be identified by a task number instead. In this embodiment all channels with signals for a specific task set the same bit in the signal register of the shell. In this way, a number of tasks equal to the number of bits in the signal register could be uniquely addressed, whereas each task may have more than one channel. The waiting is a little less specific than with unique channel identification, but the number of unnecessary wake-ups is still small and more channels can be supported with limited hardware.

Of course, the task numbers do not have to be identical to the bit numbers, (it is just simple to do it that way) as long as the relation is defined. Furthermore, it is also possible that (groups of) tasks share the same signal-interrupt-number.

Instead of identifying the task carried out by the receiver, the sender's task identification number could be signalled. In that case the receiving processor can select interrupt signals from a specific task instead of for a specific task. It may depend of the number of external tasks and the number of tasks on the processor personal preference or what seems the most useful/efficient.

These and other aspects are described in more detail in the drawing. Therein

- Figure 1 schematically shows a data processing apparatus according to the invention,

- Figure 2 schematically shows a way in which synchronization counters indicate partitions of a buffer,

5 - Figure 3 illustrates a first aspect of a synchronization method according to the invention,

- Figure 4 illustrates a second aspect of a synchronization method according to the invention,

10 - Figure 5 illustrates a further synchronization method according to the invention,

- Figure 6 illustrates in more detail a signal controller for a processor,

- Figure 7 illustrates a synchronization shell for a processor, and

- Figure 8 illustrates a channel controller.

15

Figure 1 shows a data processing apparatus comprising at least a first 1.2 and a second processing means 1.3. The first processing means, a VLIW processor 1.2 is capable of providing data by making tokens available in a buffer means, located in memory 1.5. The tokens are readable by the second processing means 1.3, a digital signal processor, for further
20 processing. The data processing apparatus further comprises a RISC processor 1.1, an ASIP 1.4, and a dedicated hardware unit 1.6. The VLIW processor 1.2, the DSP 1.3, the ASIP 1.4, the memory 1.5 and the ASIC 1.6 are mutually coupled via a first bus 1.7. The RISC processor 1.1 is coupled to a second bus 1.8 which is coupled on its turn to the first bus 1.7 via a bridge 1.9. A further memory 1.10 and peripherals 1.11 are connected to the second bus
25 1.8. The processors may have auxiliary units. For example the RISC-processor 1.1 comprises an instruction cache 1.1.1 and data cache 1.1.2. Likewise the VLIW processor has an instruction cache 1.2.1 and data cache 1.2.2. The DSP 1.3 comprises an instruction cache 1.3.1, a local memory 1.3.2, and an address decoder 1.3.3. The ASIP 1.4 comprises a local memory 1.4.1 and address decoder 1.4.2. The ASIC 1.6 comprises a local memory 1.6.1 and
30 address decoder 1.6.2. The processing means 1.2, 1.3 are each assigned a respective synchronization indicator. Both synchronization indicators are accessible by both the first 1.2 and the second processing means 1.3. The first synchronization indicator is at least modifiable by the first processing means 1.2 and readable by the second processing means

1.3. The second synchronization indicator is at least modifiable by the second processing means 1.3, and readable by the first processing means 1.2.

Each of the synchronization indicators is represented by a counter. The counter which represents the first synchronization indicator (p-counter) is indicative for a number of tokens being written by the first processing means 1.2. The counter which
 5 represents the second synchronization indicator (c-counter) is indicative for a number of tokens being read by the second processing means 1.3. Several options are possible for the skilled person to indicate the number of tokens by a counter as long as a comparison of the counter values makes it possible to calculate the number of tokens which are available to
 10 each of the processors. For example the counter value could be equal to the number of tokens mod n, wherein n is an integer value. Otherwise each step of the counter could represent a fixed number of tokens, or a token could be represented by a number of steps of the counter value.

In a practical embodiment the counters are a pointer to the address up to which
 15 the buffer means is made available to the other processor. This is schematically illustrated in Figure 2. This Figure schematically shows a buffer space 2.1 within the memory 1.5 which is used by the first processing means 1.2 for providing data to the second processing means 1.3. The buffer space 2.1 is arranged as a cyclical buffer. The buffer space 2.1 comprises a first zone 2.2 and a second zone 2.4 which contains data written by the first processing means 1.2,
 20 which is now available to the second processing means 1.3. The buffer space 2.1 further comprises a third zone 2.3 which is available to the first processing means 1.2 to write new data. The p-counter **writec** indicates the end of the first zone 2.2, and the c-counter **readc** points to the end of the second zone 2.3.

A portion 2.6 within the first zone 2.2 and the second zone 2.4 is reserved by
 25 the reservation counter **readrsv** in combination with the synchronization counter **readc**.

A portion 2.5 within the third zone 2.3 is reserved by the reservation counter **writersvc** in combination with the synchronization counter **writec**.

A subtraction of the two counters modulo the buffer size **buffsz** gives the number of valid tokens **Nc** available to the second processing means 1.3 and the number of
 30 empty tokens **Np** which are available to be filled by the first processing means 1.2.

$$(1) \quad Nc = (writec - readc) \bmod buffsz, \text{ and}$$

$$(2) \quad Np = (readc - writec) \bmod buffsz.$$

The p-counter and the c-counter are stored in a location which is accessible to at least two processing means. Each access to these counters causes a delay, as some arbitration

mechanism is required which gives access to said location. In order to further improve the efficiency the processing means are provided with a register means for locally storing an indication of the amount of tokens available. In an embodiment the first processing means 1.2 have a register for storing a counter value **readc'**, and the second processing means 1.3 have a register for storing a counter value **writec'**. The value **writec** is already available to the first processing means 1.2 as these means determine the progress of the p-counter. For analogous reasons the value **readc** is available to the second processing means. Instead of calculating the number **Nc** of actually available valid tokens the second processing means 1.3 now calculates a pessimistic estimation **Nc'** of this value according to:

$$(3) \quad \mathbf{Nc'} = (\mathbf{writec'} - \mathbf{readc}) \bmod \mathbf{buffsz}.$$

As the values **writec'** and **readc** together are indicative, i.e. a pessimistic estimation, for the number of tokens **Nc** available to be read, the facilities for locally storing these variables form register means for locally storing an indication of the amount of tokens available to be read. Alternatively the value **Nc'** may be stored locally instead of the value **writec'**.

Likewise the first processing means 1.2 now calculates a pessimistic estimation **Np'** of the actually available number of empty tokens according to:

$$(4) \quad \mathbf{Np'} = (\mathbf{readc'} - \mathbf{writec}) \bmod \mathbf{buffsz}.$$

The facilities for locally storing these variables form register means for locally storing an indication of the amount of tokens available to be written.

Alternatively the value **Np'** may be stored locally instead of the value **readc'**.

At the moment that the first processing means 1.2 detect that **Np'** has a value 0, it is necessary to update the value **readc'** in the local register of the first processing means with the momentary value **readc** of the c-counter. At the moment that the second processing means 1.3 detect that **Nc'** has a value 0, it is necessary to update the value **writec'** in the local register of the second processing means with the momentary value **writec** of the p-counter.

The data relating to the communication channel between the first 1.2 and the second processing means 1.3 may be organized in a shared data structure in addition to the information stored locally. An example of such a datastructure **CHP_channelT** as specified in the c-language is as follows:

```
typedef struct {
    int id;
    int buffsz;
    int flags;
```

```

        struct CHP_taskS* ptask;
        struct CHP_taskS* ctask;
        union {
            CHP_bufferT* buffer_pointers;
5           struct {
                int token_size;
                CHP_bufferT buffer;
            } buffer_data;
        } channel;
10 unsigned writec;
    unsigned readc;
    CHP_channel_hwT* pchan;
    CHP_channel_hwT* cchan;
    } CHP_channelT;
15

```

Apart from the counter values **writec** and **readc** this data structure comprises the following data.

id is a value identifying the channel, so as to enable a processing scheme including a plurality of channels, for example a first channel for transferring data from a first processing means to a second processing means, a second channel for transferring data from the second processing means to the first processing means and a third channel for transferring data from the second processing means to a third processing means.

The value **buffsz** indicates the size of the buffer, i.e. as the number of tokens which can be stored in the buffer.

The value **flags** indicates properties of the channel, e.g. if the synchronization is polling or interrupt based, and whether the channel buffers are allocated directly or indirectly. As an alternative it can be decided to give the channel predetermined properties, e.g. restrict the implementation to interrupt based synchronization with directly allocated buffers. In that case the value flags may be omitted.

ptask and **ctask** are pointers to the structure describing the task of the first processing means, the producer, and the task of the second processing means, the consumer.

The task structure may contain for example
 - an identifier for the task (whose task structure is it)

- a function pointer (if it is a task on the embedded processor; then after booting the root_task can jump to this function and start the application. Void otherwise.

- a device type: to indicate on what type of device the task should be running.

5 This is useful for the boot process: a task running as a Unix process has to be initialized in a different way than a task running on a DSP or on a Embedded processor or on dedicated hardware. By having a major device type, it is easy to select the proper boot procedure.

- a device number: This enable to distinguish between e.g. the first Unix co-processor from the second. this can be done by giving them a unique number.

10 - the number of channels

- a list of pointers to channel datastructures.

In this way, when e.g a UNIX task is started, it can first read its task structure, and then read all the information about all the channels connected to that task. This makes the booting process a lot easier, and avoids that very time some task is added, a control process
15 has to be modified to have all tasks load the proper data structures.

The union **channel** indicates the location of the buffer. Either the buffer is located indirectly via a pointer **buffer_pointers** to the structure **CHP_bufferT**, or the buffer is included in the structure **CHP_channelT**.

20 The integer **token_size** indicates the size of the tokens which are exchanged by the producer and the consumer, e.g. in the number of bytes.

As described above it is favorable if the processing means comprise local data such as the counter value **writec'** for the consuming task. Preferably the data structure **CHP_channelT** comprises references **pchan**, **cchan** to a datastructure comprising the local task information.

25 Such a datastructure may have the following form specified in the language C:

```
typedef struct {
    unsigned sgnl_reg_addr;
    unsigned sgnl_value;
    30 unsigned rsmpr_addr;
    int buffsz;
    CHP_bufferT buf_ptr;
    unsigned lsmpr_reg;
    int in_out;
```

```

        int token_size;
    } CHP_channel_hwT;

```

The structure comprises an unsigned integer **sgnl_reg_addr** indicating the signal register address of other device with which the processing means is communicating. An interrupting processor may leave in the signal register of a device an indication of the task or of the channel for which the interrupt took place.

It further comprises unsigned integer **sgnl_value** indicating its own signaling value.

The unsigned value **rsmpr_addr** indicates the remote synchronization counter address in the other device.

As described above, the buffersize **buffsz** is used by the producer to calculate the number of empty tokens available, and by the consumer to calculate the number of written tokens available.

The value **buf_ptr** indicates the base address of the buffer.

The unsigned integer **lsmpr_reg** stores the value of the local channel synchronization counter.

The type of channel input/output is determined from the integer **in_out**.

The integer **token_size** indicates the size of the tokens which is exchanged via the channel.

Figure 3 illustrates a first aspect of a method according to the invention of synchronizing a first and a second processing means in a data processing apparatus.

In step 3.1 the first processing means 1.2 generates one or more tokens, In step 3.2 the first processing means reads the first counter **writec** which is indicative for a number of tokens made available to the second processing means.

In step 3.3 the first processing means 1.2 read the second counter **readc** which is indicative for the number of tokens consumed by the second processing means 1.3.

In step 3.4 the first processing means compares these counters by means of the calculation of equation 2.

In step 3.5 the first processing means 1.2 decide in dependence of this comparison either to carry out steps 3.6 and 3.7, if the value of **Np** is greater or equal than the number of tokens generated, or to carry out step 3.8 in the other case.

In step 3.6 the first processing means 1.2 writes the tokens to the buffer means 2.1 and subsequently modifies the first counter **writec** in step 3.7, after which it continues with step 3.1.

5 In step 3.8 the first processing means wait, e.g. for a predetermined time interval, or until it is interrupted and repeats steps 3.2 to 3.5.

The second processing means 1.3 carries out an analogous procedure, as illustrated in Figure 4.

10 In step 4.2 the second processing means 1.3 reads the first counter **writec** which is indicative for a number of tokens made available to it by the first processing means 1.2.

In step 4.3 the second processing means 1.3 read the second counter **readc** which is indicative for the number of tokens it has consumed.

In step 4.4 the second processing means 1.3 compares these counters by means of the calculation of equation 1.

15 In step 4.5 the second processing means 1.3 decide in dependence of this comparison either to carry out steps 4.6 and 4.7, if the value of **Nc** is greater or equal than the number of tokens generated, or to carry out step 4.8 in the other case.

20 In step 4.6 the second processing means 1.3 reads the tokens from the buffer means and subsequently modifies the second counter in step 4.7. Before continuing with step 4.1 it may execute a data processing step 4.9.

In a preferred embodiment a processing means locally stores a copy of a value of the other processing means with which it is communicating.

Figure 5 illustrates how this local copy is used in a preferred method according to the invention.

25 Steps 5.1 and 5.2 are analogous to steps 3.1 and 3.2 in Figure 3.

However in step 5.3 instead of reading the value **readc** of c-counter, which is stored remotely, the first processing means 1.2 read a locally stored value **readc'**. This read operation usually takes significantly less time than reading the remote value **readc**.

30 Step 5.4 is analogous to step 3.4 in Figure 3, apart from the fact that the first processing means 1.2 use this locally stored value to calculate **Np'**, as in equation 4.

In step 5.5 the first processing means 1.2 decide in dependence of this comparison either to carry out steps 5.6 and 5.7, if the value of **Np** is greater or equal than the number of tokens generated, or to carry out steps 5.8, 5.10 and 5.11 in the other case.

Steps 5.6 and 5.7 are analogous to steps 3.6 and 3.7 of Figure 3.

In step 5.8 the first processing means may wait, e.g. for a predetermined time interval, or until it is interrupted. Subsequently it reads the remote value **readc** in step 5.10 and store this value locally as the variable **readc'** in step 5.11. According to this method it is only necessary to read the remote value **readc** if the number of empty tokens calculated from the local stored value **readc'** is less than the number of tokens which is to be written in the buffer. The value **Np'** could be stored locally instead of the value of **readc'**. In this case the value **Np'** should be updated after each write operation for example simultaneously with step 5.7.

Likewise it is possible to improve the efficiency of the second processing means, executing the consuming process, by using a locally stored value of **prodc** or **Nc'**.

In order to further reduce communication overload, the processing means 6.1 may be provided with a signal controller 6.2 as is schematically illustrated in Figure 6. The signal controller comprises a signal register 6.3 and a mask register 6.4. The contents of the registers in the signal controller are compared to each other in a logic circuit 6.5 to determine whether the processor 6.1 should receive an interrupt. Another processor sending the processor a message that it updated a synchronization counter updates the signal register 6.5 so as to indicate for which task it updated this counter. For example, if each bit in the signal register represents a particular task, the message has the result that the bit for that particular task is set. On the other hand the processor 6.1 indicates in the mask register 6.4 for which tasks it should be interrupted. The logic circuit 6.5 then generates an interrupt signal each time that a message is received for one of the tasks selected by the processor 6.1. In the embodiment shown the logic circuit 6.5 comprises a set of AND-gates 6.5.1-6.5.n, each AND gate having a first input coupled to a respective bit of the signal register 6.3 and a second input coupled to a corresponding bit of the mask register 6.4. The logic circuit 6.5 further comprises an OR-gate 6.5.0. Each of the AND-gates has an output coupled to an input of the OR-gate. The output of the OR-gate 6.5.0 provides the interrupt signal.

Figure 7 shows an embodiment wherein the processor 7.1 has a separate synchronization shell 7.2 for supporting communication with other processing means via a communication network, e.g. a bus 7.3. The synchronization shell 7.2 comprises a bus adapter 7.4, a signal register 7.5 for storing the identity of tasks for which the synchronization shell 7.2 has received a message. The synchronization shell 7.2 further comprises channel controllers 7.6, 7.7. These serve to convert commands of the processor 7.6 in signals to the bus 7.3. Usually an application specific device 7.1 will execute less tasks in parallel than is

the case for a programmable processor 6.1. Consequently it is less important to apply interrupt selection techniques as illustrated in Figure 6.

Figure 8 shows a channel controller 8.1 in more detail. The channel controller 8.1 comprises a generic bus master slave unit 8.2, a register file 8.3 and a control unit 8.4.

5 The bus adapter 7.4 and the generic bus master slave unit 8.2 together couple the channel controller 8.1 to the bus. The bus adapter 7.4 provides an adaptation from a particular interconnection network, e.g. a PI-bus or an AHB-bus to a generic interface. The generic bus master slave unit 8.2 provides for an adaptation of the synchronization signals to said generic interface. In this way it is possible to support different channel controller types and different buses with a relatively low number of different components.

The register file 8.3 stores the synchronization information.

In case the device synchronization interface of a processor 7.1 issues the signal **Claim** in order to claim a number of writable or readable tokens in the buffer, the control unit 8.4 verifies whether this number is available by comparing the locally stored value of the remote counter **remotec** with its reservation counter **localrsvc**. The notation **remotec** signifies **writetc** for an input channel and **readc** for an output channel. The notation **localrsvc** refers to **readrsvc** for an input channel and **writersvc** for an output channel.

If the verification is affirmative, the address of a token **Token Address** is returned. Otherwise, the upper boundary address of the buffer space reserved for the processor 7.1 could be returned. The signal **Token Valid** indicates if the claim for tokens was acknowledged, and the processor's synchronization interface can rise the signal **Claim** again. In this way a token address can be provided to the processor at each cycle. If the outcome of the first verification is negative, the channel controller 8.1 reads the remote counter indicated by the address **remotecaddr** and replaces the locally stored value **remotec** by the value stored at that address. The control unit 8.4 now again verifies whether the claimed number of tokens is available.

If the request fails, the channel controller 8.1 could either poll the remote counter regularly in a polling mode or wait for an interrupt by the processor with which it communicates in an interrupt mode. In the mean time it may proceed with another task. The variable **inputchannel** in the register indicates to the channel controller whether the present channel is an input or an output channel and which of these modes is selected for this channel.

After a successful claim the variable **localrsvc** is updated in conformance with the number of tokens that was claimed.

Instead of the variable **remotec**, the register file could comprise a variable indicating the number of available tokens calculated with the last verification.

In case that the processor 7.1 signals **Release_req** the local counter **localc** is updated in accordance with this request. This local counter **localc** is **readc** for an input channel and **writec** for an output channel. Optionally the signal **Release_req** may be kept high so that the processor 7.1 is allowed to release tokens at any time. However, this signal could be used to prevent flooding the controller when it is hardly able to access the bus.

Alternatively the synchronization process could be implemented in software by using a claim and a release function. By executing the claim function a processor claims a number of tokens for a particular channel and waits until the function returns with the token address. By executing the release function the processor releases a number of tokens for a particular channel. Separate functions could exist for claiming tokens for writing or tokens for reading. Likewise separate functions may be used for releasing.

It is remarked that the scope of protection of the invention is not restricted to the embodiments described herein. Neither is the scope of protection of the invention restricted by the reference numerals in the claims. The word 'comprising' does not exclude other parts than those mentioned in a claim. The word 'a(n)' preceding an element does not exclude a plurality of those elements. Means forming part of the invention may both be implemented in the form of dedicated hardware or in the form of a programmed general purpose processor. The invention resides in each new feature or combination of features.

CLAIMS:

1. A data processing apparatus comprising at least a first and a second processing means, the first processing means being capable of providing data by writing tokens in a buffer means which are readable by the second processing means for further processing, the processing means being assigned a first and a second synchronization indicator, the first
5 synchronization indicator being modifiable by the first processing means and being readable by the second processing means, and the second synchronization indicator being modifiable by the second processing means, and readable by the first processing means, characterized in that, each of the synchronization indicators is represented by a counter, the counter which
10 represents the first synchronization indicator being indicative for a number of tokens being written by the first processing means, and the counter which represents the second synchronization indicator being indicative for a number of tokens being read by the second processing means.

15 2. Data processing apparatus according to claim 1, wherein at least one of the processors locally maintains a reservation counter, wherein a first command issued by a processor results in a verification whether a number of tokens is available to it by comparing its reservation counter with the synchronization counter maintained by the other processor, the reservation counter being updated if the outcome of the verification is affirmative.

20 3. A data processing apparatus according to claim 1, characterized in that the counters have a range which is larger than the total number of tokens which is contained in a full buffer means.

25 4. A data processing apparatus according to claim 1, characterized in that at least one of the processing means is a dedicated processor.

5. A data processing apparatus according to claim 1, characterized in that at least one of the processing means is a computer program executed by a general purpose processor or an application specific programmable device.

5 6. A data processing apparatus according to claim 5, characterized in that a first processing means gets access to an empty token by means of a producer-claim call, and releases the filled token with a producer-release call.

7. A data processing apparatus according to claim 5, characterized in that a
10 second processing means claims a filled token, with a consumer-claim call, and releases the token with a consumer-release call.

8. A data processing apparatus according to one of the previous claims,
characterized in that the first processing means comprises a register means for locally storing
15 an indication of the amount of tokens available to be written.

9. A data processing apparatus according to one of the previous claims,
characterized in that the second processing means comprises a register means for locally
storing an indication of the amount of tokens available to be read.

20

10. A method of synchronizing a first and a second processing means in a data processing apparatus, by means of a first and a second synchronization indicator, the first processing means being capable of providing data by writing tokens via a buffer means to the second processing means for further processing, wherein

- 25 a. the first processing means generates one or tokens,
b. compares the first and the second synchronization indicator,
c. in dependence of this comparison either
c1. writes the tokens to the buffer means and modifies the first synchronization indicator, after which it continues with step a
30 c2. repeats steps b-c, and wherein
d. the second processing means compares the first and the second synchronization indicator,
e. in dependence of this comparison either

e1. reads the tokens from the buffer means and modifies the second synchronization indicator,

e2. repeats steps d,e,

characterized in that the first processing synchronization indicators are counters, the counter which represents the first synchronization indicator being indicative for a number of tokens available to the second processing means, and the counter which represents the second synchronization indicator being representative for an amount of space available for writing tokens by the first processing means in the buffer means, wherein a modification of the synchronization means is an incrementation of the corresponding counter.

1/6

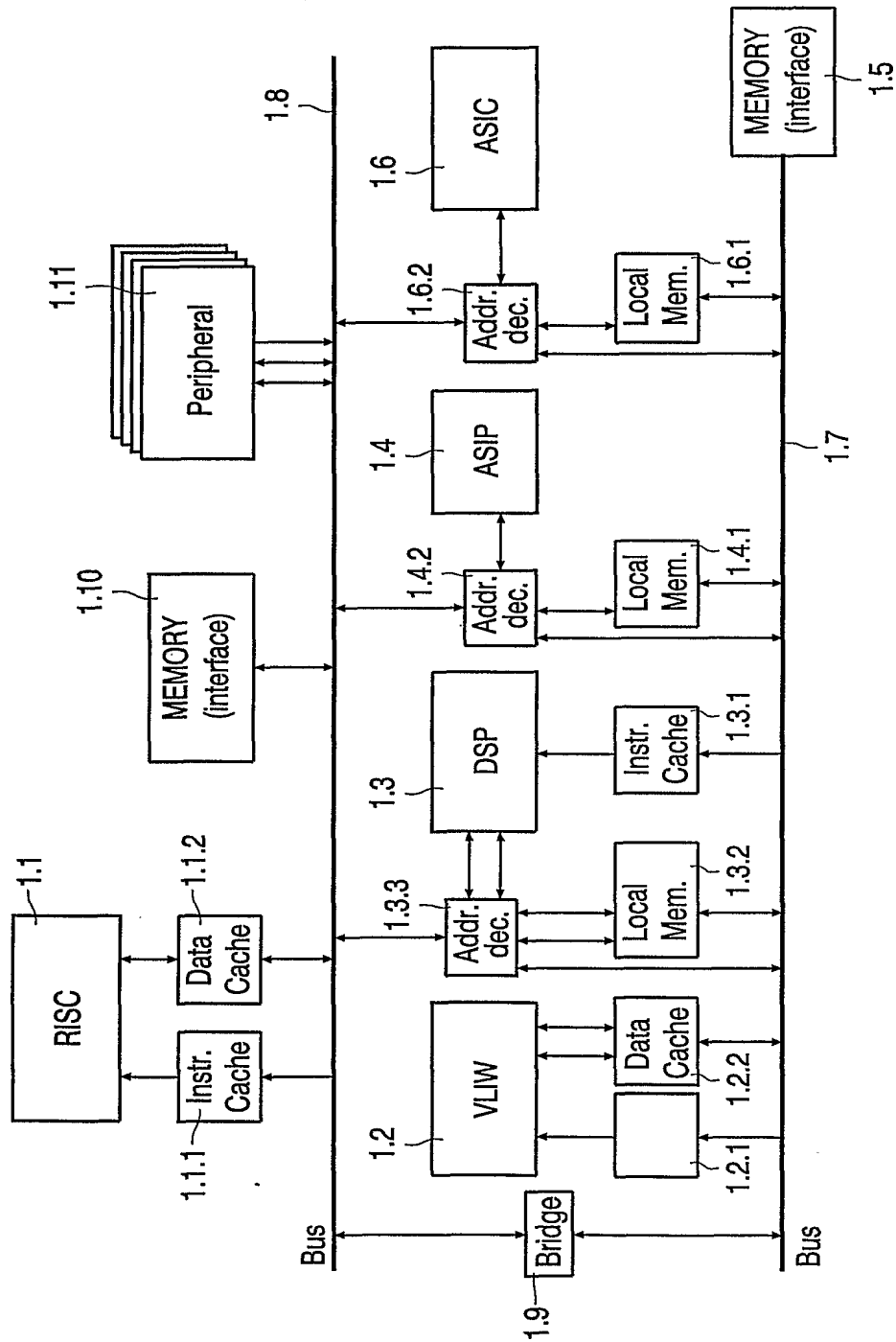


FIG. 1

2/6

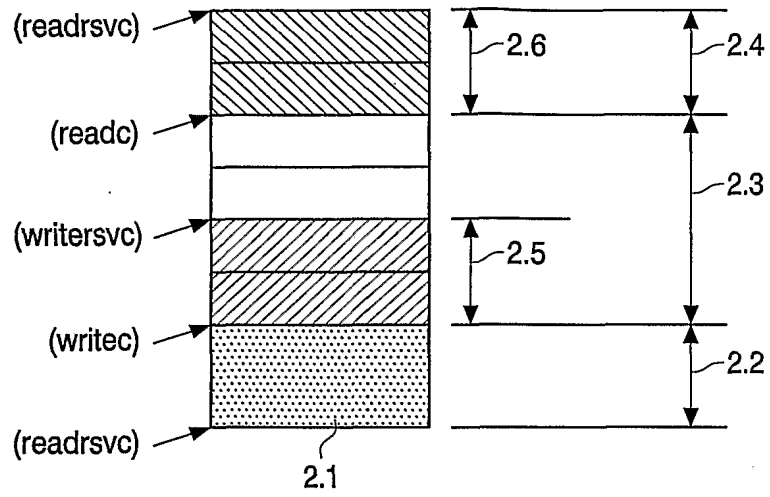


FIG. 2

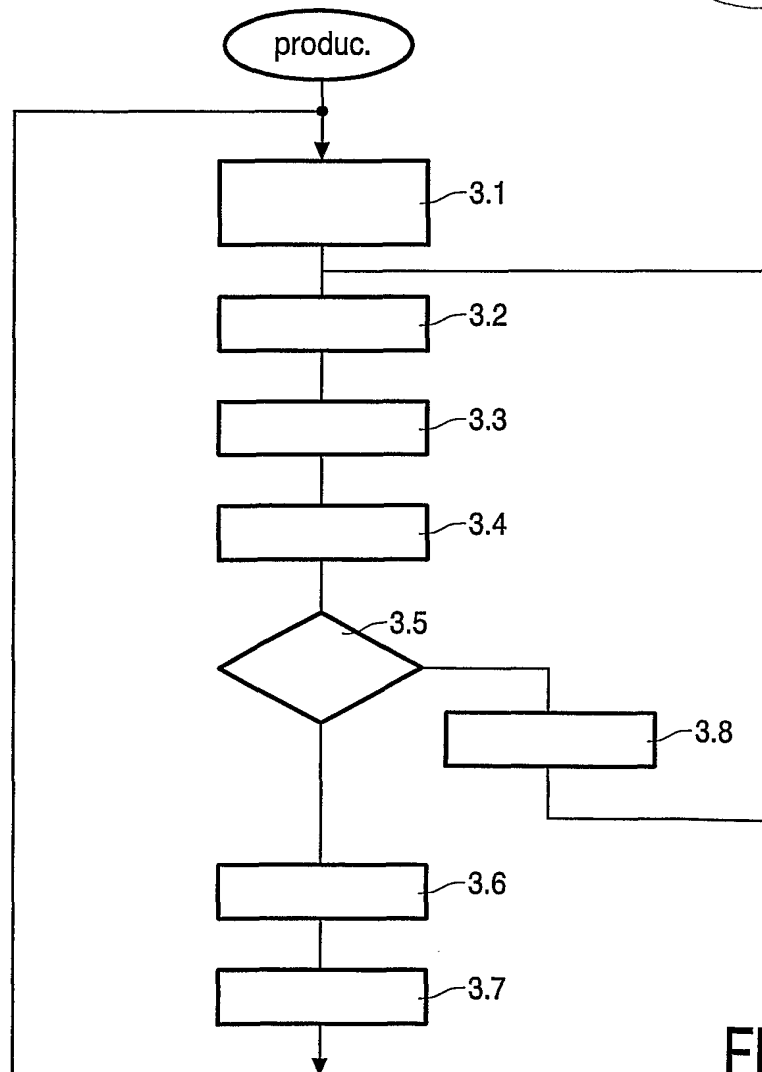


FIG. 3

3/6

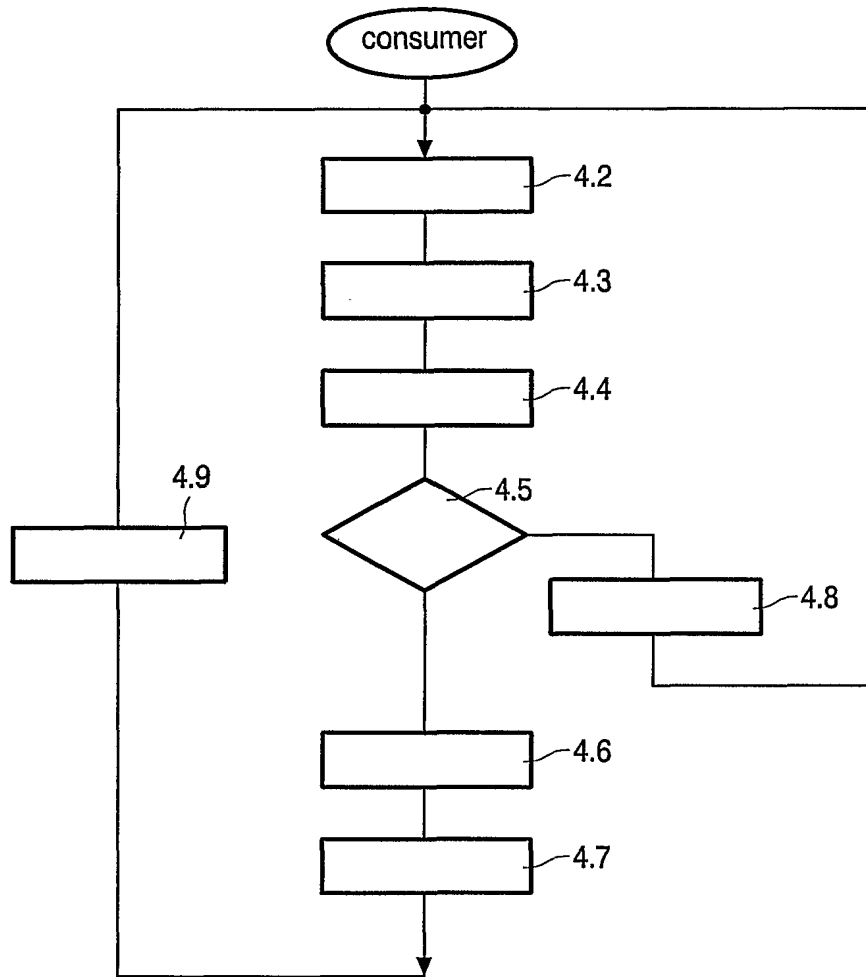


FIG. 4

4/6

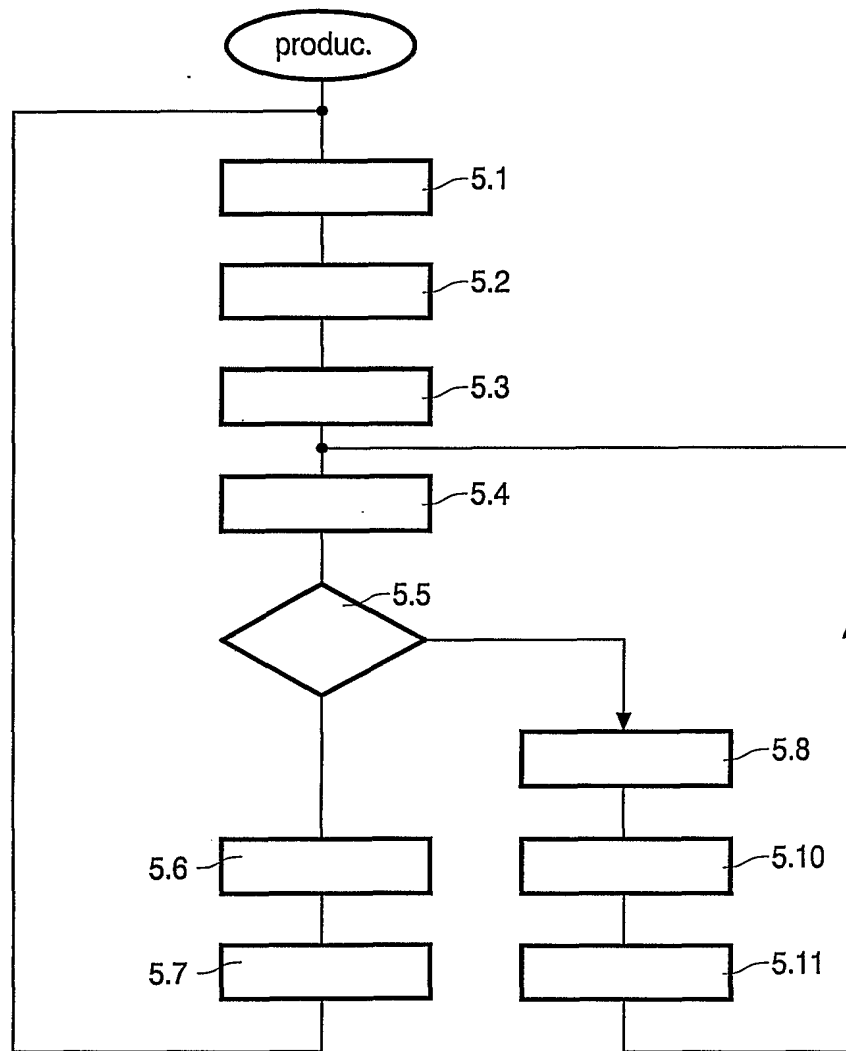


FIG. 5

5/6

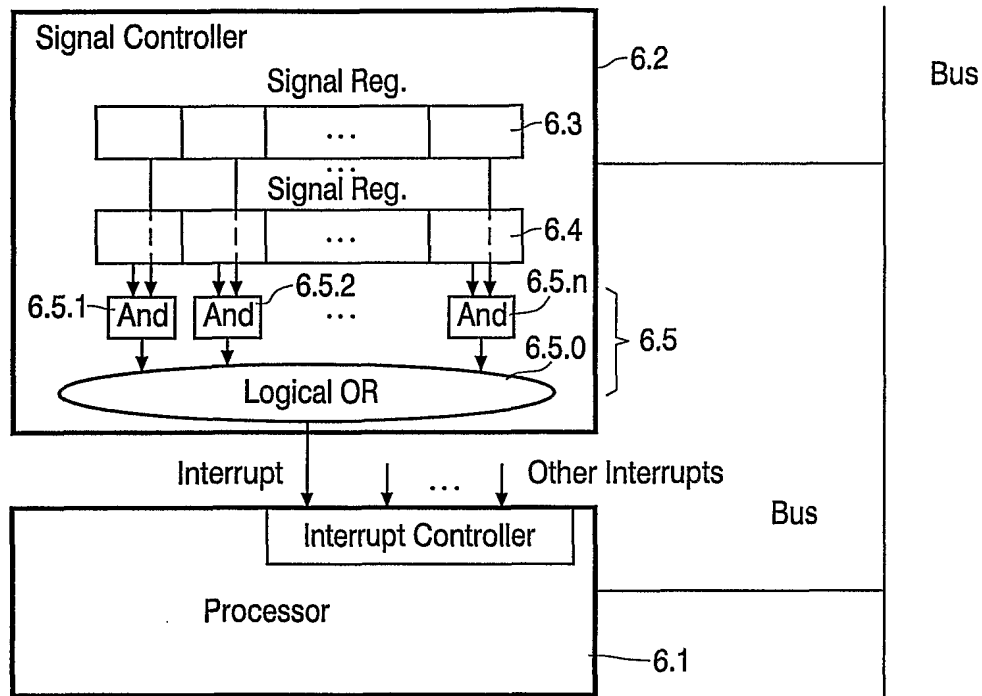


FIG. 6

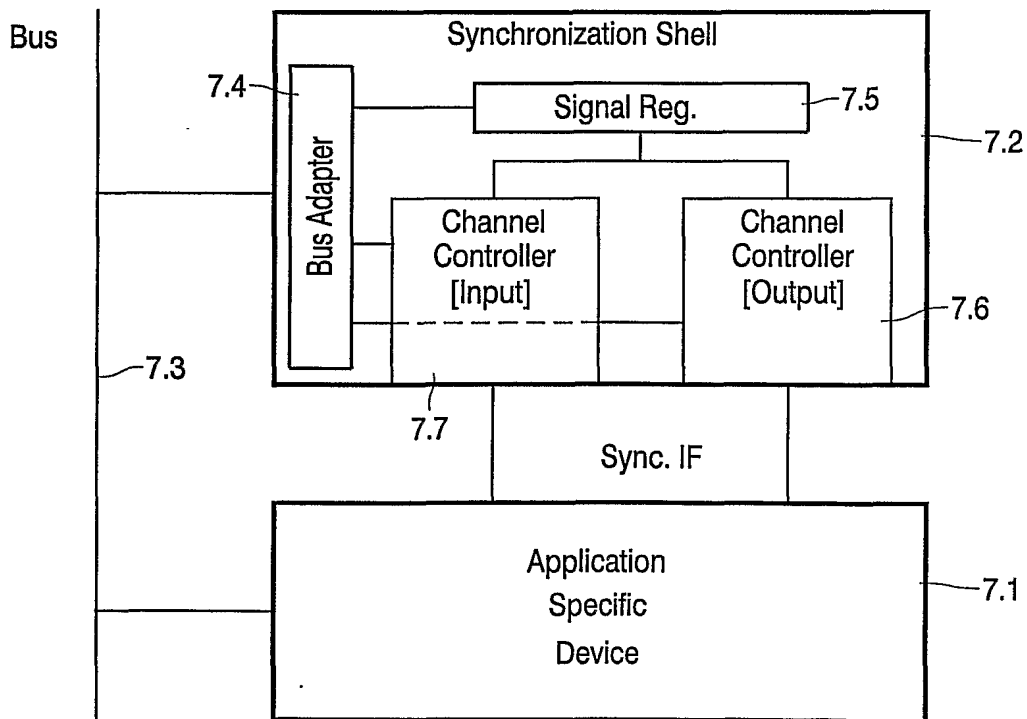


FIG. 7

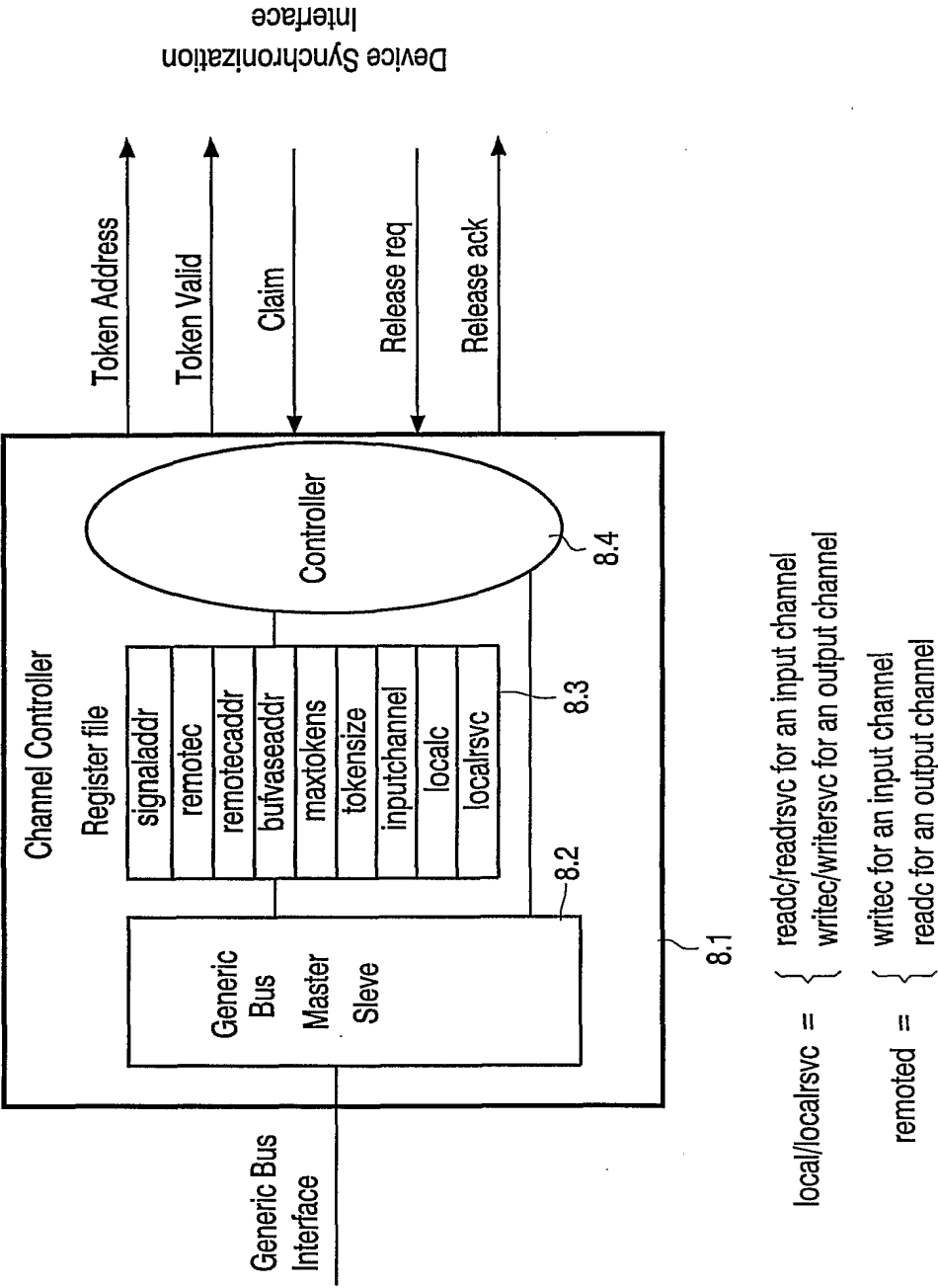


FIG. 8